Init

In [1]:

```
try:
    import os
    import glob
    import sys
    import math
    from typing import List, Optional
    from functools import partial
    import itertools
    import copy
except Exception as e:
    print(e)
    print("Some of the libraries needed to run this script were not installed or were not loaded. Please install the libraries befo
re proceeding.")
```

In [2]:

```
sys.path.append(os.environ['DEV_AUTOTS'])
sys.path.append(os.environ['CAPSTONE_PYTHON_SOURCE'])
folder = os.environ['CAPSTONE_DATA']
```

In [3]:

```
try:
    # Data Tables
    import pandas as pd
    import numpy as np

    # Plotting
    import matplotlib.pyplot as plt
    import plotly.offline as py
    from plotly.offline import plot
    py.init_notebook_mode(connected=True)

    # EDA and Feature Engineering
    from scipy.spatial.distance import euclidean, pdist, squareform
    import statsmodels.api as sm

    # Auto Time Series
    import auto_ts as AT

    # Optimizer
    from skopt import gp_minimize
    from skopt.space import Real, Integer
    from skopt.plots import plot_convergence
except Exception as e:
    print(e)
    print("Some of the libraries needed to run this script were not installed or were not loaded. Please install the libraries befo
re proceeding.")
```

```
Running Auto Timeseries version: 0.0.24
```

In [4]:

```
%load_ext autoreload
%autoreload 2
```

In [5]:

```
try:
    from ETL.ETL import loadDataset, getTopProducts
    from similarity.similarity import mergeTopSimilar, loadSimilarity
    from charting.charting import surface3DChart
except Exception as e:
    print(e)
    print("Some of the libraries needed to run this script were not installed or were not loaded. Please install the libraries befo
re proceeding.")
```

In [6]:

```
dataRaw= loadDataset(version=4)
```

Prep Data

In [7]:

```
#Parameters
#ChainMaster = 'SPECS'
#ProdCat='SUP PREM WHISKEY'
TOP_PRODUCTS = 3  # How many products to consider in the category
TOP_SIMILAR = 3  # Get TOP_SIMILAR most similar products

LOG_TRANSFORM = True # Take log of 9L cases to smooth out peaks and valleys
ZERO_ADDER = 0.1

RESAMPLE_FREQ = 'M'

# Pricing changes every 4 weeks
if RESAMPLE_FREQ == 'M':    FORECAST_PERIOD = 1
if RESAMPLE_FREQ == 'W':    FORECAST_PERIOD = 4
if RESAMPLE_FREQ == '2W':   FORECAST_PERIOD = 2

# Seasonal Period
if RESAMPLE_FREQ == 'M':    SEASONAL_PERIOD = 12  # Yearly
if RESAMPLE_FREQ == 'W':    SEASONAL_PERIOD = 13 # Quarterly (we can also take yearly = 52, but SARIMAX becomes too slow)
if RESAMPLE_FREQ == '2W':   SEASONAL_PERIOD = 13 # This becomes problematic --> for quarterly, should we take 6 biweekly periods or 7 bi-weekly periods. Instead I just took half yearly period

print("="*50)
print("Parameters being used...")
print("="*50)
print(f"Resample Frequency = {RESAMPLE_FREQ}")
print(f"Forecast Period = {FORECAST_PERIOD}")
print(f"Seasonal Period = {SEASONAL_PERIOD}")
```

```
==================================================
Parameters being used...
==================================================
Resample Frequency = M
Forecast Period = 1
Seasonal Period = 12
```

Model Flow

Functions

Core Functions

In [8]:

```
COL_TIME = 'WeekDate'
COL_PREDS = ['9L Cases'] #Demand
COL_PRICE= ['Dollar Sales per 9L Case'] #Price

def modelsLoadData(ProductsList,dataRaw,ChainMaster):
    all_data = []

    if(ChainMaster!=''):
        dfSimilarity = loadSimilarity(version=4)
    else:
        dfSimilarity = loadSimilarity(version=4,allCustomers=True)

    for i, Product in enumerate(ProductsList):
        (dataModel,colExog,colEnc,colDec) = mergeTopSimilar(dataRaw, dfSimilarity
                                                            ,ChainMaster=ChainMaster
                                                            ,Product=Product
                                                            ,ProductsList=ProductsList
                                                            ,topn=TOP_SIMILAR
                                                            ,periodCol = COL_TIME
                                                            ,resampleFreq=RESAMPLE_FREQ
                                                            ,encodeCols=True)


        if i == 0: print(f"Decoder: {colDec}")

        print("\n\n")
        print("-"*50)
        print(f"Product: {colDec.get(str(i))}")
        print("-"*50)

        #colExog = colExog + colEndog
        print(f"Exogenous Price Columns: {colExog}")

        allCols=[COL_TIME]+COL_PREDS+ colExog
        data=dataModel[allCols]
        print(f"% of weeks without a purchase: {sum(data['9L Cases'] == 0)/data.shape[0]*100}")
        all_data.append(data)

    all_data_non_transformed =  copy.deepcopy(all_data)

    if LOG_TRANSFORM:
        print("Log Transforming")
        for i in np.arange(len(all_data)):
            all_data_non_transformed[i] = all_data[i].copy(deep=True)
            all_data[i][COL_PREDS] = np.log10(all_data[i][COL_PREDS] + ZERO_ADDER)
```

```
            print(f"\tProduct: {colDec.get(str(i))}")
    return(all_data,all_data_non_transformed,colExog,colEnc,colDec)

def ModelsWhiteNoise(all_data)           :
    ## WHITE NOISE TEST
    white_noise_all = []
    white_noise_df_all = []
    #check if there are 12, 24, 48 data points
    for i, data in enumerate(all_data):
        lags=[12,24,48]
        lags=[x  for x in lags if x < data.shape[0]]
        white_noise_df = sm.stats.acorr_ljungbox(data[COL_PREDS], lags=lags, return_df=True)
        white_noise_df_all.append(white_noise_df)
        if any(white_noise_df['lb_pvalue'] > 0.05):
            white_noise = True
        else:
            white_noise = False
        white_noise_all.append(white_noise)

        print(white_noise_df)
        print(f"\nIs Data White Noise: {white_noise}")

    return(white_noise_all)

def ModelsTestTrain(all_data,all_data_non_transformed):
    all_train = []
    all_test = []

    all_train_non_transformed = []
    all_test_non_transformed = []

    for i, data in enumerate(all_data):
        train = all_data_non_transformed[i].iloc[:-FORECAST_PERIOD]
        test = all_data_non_transformed[i].iloc[-FORECAST_PERIOD:]
        all_train_non_transformed.append(train)
        all_test_non_transformed.append(test)

        train = data.iloc[:-FORECAST_PERIOD]
        test = data.iloc[-FORECAST_PERIOD:]
        all_train.append(train)
        all_test.append(test)

        print(train.shape,test.shape)
    return(all_train,all_test,all_train_non_transformed,all_test_non_transformed)

def ModelsFit(all_data,all_train,all_test,withSimilar,model_type=['SARIMAX','ML','prophet','auto_SARIMAX']):
    from joblib import Parallel, delayed
```

```python
    def modelsFun(i):
        train = all_train[i]
        test = all_test[i]
        import auto_ts as AT
        if(withSimilar==False):
            train = train[train.columns[0:3]] #3rd col has the curr product price
        print(train.columns)

        automl_model = AT.AutoTimeSeries(
            score_type='rmse', forecast_period=FORECAST_PERIOD, # time_interval='Week',
            non_seasonal_pdq=None, seasonality=True, seasonal_period=SEASONAL_PERIOD,
            model_type=model_type,
            verbose=0)

        #colP = COL_PREDS[COL_PREDS in train.columns]
        automl_model.fit(train, COL_TIME, COL_PREDS, cv=10, sep=',') #cv=10
        return(automl_model)

    args = np.arange(len(all_data))

    all_models = Parallel(n_jobs=-1, verbose=1
                          #, backend="threading"
                           , backend="loky"
                         )(
             map(delayed(modelsFun), args))


    return(all_models)

def get_rmse(predictions, targets):
    return np.sqrt(((np.array(predictions) - np.array(targets)) ** 2).mean())

def modelNaive(all_data,all_train,all_test,all_train_non_transformed,season=12,windowLength=8):
    from sktime.forecasting.naive import NaiveForecaster
    import statistics
    from tscv import GapWalkForward # type: ignore
    all_naives=pd.DataFrame(columns=['ID','Best Type','Best RMSE'])
    types=['last','seasonal_last','mean']
    #add window code

    NFOLDS=5
    for i, data in enumerate(all_data):
        yTrain = pd.Series(all_train[i][COL_PREDS[0]])
        yTest = pd.Series(all_test[i][COL_PREDS[0]])
        yTrain = yTrain.append(yTest) # merging as we are gong to do cv
        rmses=[]
        naive_models=[]
        for t in types:
```

```python
            #naive_forecaster = NaiveForecaster(strategy="last")
            cv = GapWalkForward(n_splits=10, gap_size=0, test_size=FORECAST_PERIOD)
            cvRmse=[]
            for fold_number, (train, test) in enumerate(cv.split(yTrain)):
                cv_train = yTrain.iloc[train]
                cv_test = yTrain.iloc[test]

                naive_forecaster = NaiveForecaster(strategy=t,sp=season,window_length=windowLength)
                naive_forecaster.fit(cv_train)
                yPred = naive_forecaster.predict(np.arange(len(cv_test)))
                rmse=get_rmse(yPred, cv_test)
                cvRmse.append(rmse)
            #naive_models.append(naive_forecaster) #last forecaster
            rmses.append(np.mean(cvRmse))
        bestRmse = np.argmin(rmses)
        bestModel = NaiveForecaster(strategy=types[bestRmse],sp=season)
        yTrainNonTrasformed = pd.Series(all_train_non_transformed[i][COL_PREDS[0]])
        bestModel.fit(yTrainNonTrasformed)
        all_naives=all_naives.append(
            {'ID':i
             ,'Best Type': types[bestRmse]
             ,'Best RMSE': rmses[bestRmse]
             ,'Best Naive': bestModel
             ,'All Types': [types]
             ,'All RMSEs': [rmses]
             ,'All Naives':naive_models
            }
            ,ignore_index=True)
    print(all_naives)
    return(all_naives)

def centerLog(text,w,pre='\n',post=''):
    t=int((w-len(text))/2-1)
    return(pre+'='*t+' '+text+' '+'='*(w-len(text)-t-2)+post)

def printLog(main,subs,linesPre=2,linesPost=1):
    import datetime
    if(isinstance(subs,list)== False): subs=[subs]
    maxw=max([len(x) for x in [main] + subs])+10
    print("\n"*linesPre
          +"="*maxw+" ("+str(datetime.datetime.now())+")"
          +centerLog(main,maxw)
          +''.join([centerLog(x,maxw) for x in subs])
          +"\n"+"="*maxw
          +"\n"*linesPost
         )
```

Call Function

In [9]:

```
def runModels(ProductsList,dataRaw,ChainMaster):
    printLog("GET DATA",ChainMaster)
    all_data,all_data_non_transformed,colExog,colEnc,colDec = modelsLoadData(ProductsList,dataRaw,ChainMaster)

    printLog("WHITE NOISE",ChainMaster)
    white_noise = ModelsWhiteNoise(all_data)

    printLog("TEST/TRAIN",ChainMaster)
    all_train, all_test,all_train_non_transformed,all_test_non_transformed = ModelsTestTrain(all_data,all_data_non_transformed)

    all_stats = pd.DataFrame()
    all_stats['Product'] = ProductsList
    all_stats['Chain Master'] = ChainMaster
    all_stats['White Noise'] = white_noise

    printLog("NAIVE",ChainMaster)
    naive = modelNaive(all_data,all_train,all_test,all_data_non_transformed,season=4,windowLength=8)
    all_stats['Naive Best Type'] = [naive.iloc[x]['Best Type'] for x in np.arange(len(all_data)) ]
    all_stats['Naive Best RMSE'] = [naive.iloc[x]['Best RMSE'] for x in np.arange(len(all_data)) ]
    all_stats['Naive Best Model'] = [naive.iloc[x]['Best Naive'] for x in np.arange(len(all_data)) ]

    printLog("Multivar P0",ChainMaster)
    multivarP0 = ModelsFit(all_data,all_train,all_test,withSimilar = False)
    all_stats['P0 Best Model Name'] = [multivarP0[x].get_leaderboard().iloc[0]['name'] for x in np.arange(len(all_data)) ]
    all_stats['P0 Best Model RMSE'] = [multivarP0[x].get_leaderboard().iloc[0]['rmse'] for x in np.arange(len(all_data)) ]
    all_stats['P0 Best Model'] = multivarP0 #[multivarP0[x] for x in np.arange(len(all_data)) ]

    printLog("Multivar P0+Sim",ChainMaster)
    multivarP0Sim = ModelsFit(all_data,all_train,all_test,withSimilar = True )
    all_stats['P0+Sim Best Model Name'] = [multivarP0Sim[x].get_leaderboard().iloc[0]['name'] for x in np.arange(len(all_data)) ]
    all_stats['P0+Sim Best Model RMSE'] = [multivarP0Sim[x].get_leaderboard().iloc[0]['rmse'] for x in np.arange(len(all_data)) ]
    all_stats['P0+Sim Best Model'] = multivarP0Sim #[multivarP0Sim[x] for x in np.arange(len(all_data)) ]

    return(all_stats)
```

Loop

In [10]:

```
ChainMasters =  [''] +  dataRaw['Chain Master'].unique().tolist()
ProdCats = dataRaw['Category (CatMan)'].unique().tolist()
display(ChainMasters,ProdCats)
```

```
['', 'THE BARREL HOUSE', 'WESTERN BEV LIQ TX', 'SPECS']
```

```
['ECONOMY VODKA', 'SUP PREM WHISKEY']
```

Testing Models

In [12]:

```
#getting train test
if False:
    ChainMaster=ChainMasters[0]
    ProductsList = getTopProducts(dataRaw, ChainMaster='WESTERN BEV LIQ TX', ProdCat='SUP PREM WHISKEY', topN=TOP_PRODUCTS, timeCol
='WeekDate')
    all_data,all_data_non_transformed,colExog,colEnc,colDec = modelsLoadData(ProductsList,dataRaw,ChainMaster)
    all_train, all_test,all_train_non_transformed,all_test_non_transformed = ModelsTestTrain(all_data,all_data_non_transformed)
```

```
resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1.75L', '1': 'JACK DANIELS BLK WHSKY  750M', '2': 'JACK DANIELS BLK WHSKY  1
L'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['0', '2', '1']
% of weeks without a purchase: 1.1904761904761905
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['1', '2', '0']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1']
% of weeks without a purchase: 0.0
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS BLK WHSKY  1L
(83, 5) (1, 5)
(83, 5) (1, 5)
(83, 5) (1, 5)
```

In [14]:

```
#Fitting model
if False:
    i=1
    withSimilar=False

    train = all_train[i]
    test = all_test[i]
    import auto_ts as AT
    if(withSimilar==False):
        train = train[train.columns[0:3]] #3rd col has the curr product price
    print(train.columns)
    #model_type=['SARIMAX','ML','prophet','auto_SARIMAX']
    model_type=['prophet']
    automl_model = AT.AutoTimeSeries(
        score_type='rmse', forecast_period=FORECAST_PERIOD, # time_interval='Week',
        non_seasonal_pdq=None, seasonality=True, seasonal_period=SEASONAL_PERIOD,
        model_type=model_type,
        verbose=0)

     #colP = COL_PREDS[COL_PREDS in train.columns]
    automl_model.fit(train, COL_TIME, COL_PREDS, cv=1, sep=',') #cv=10
```

In [15]:

```
#prediction
if False:
    display(automl_model.get_leaderboard())
    df=pd.DataFrame({'WeekDate': [pd.to_datetime('2019-12-31')],'0':[266.51]})
    prediction=automl_model.predict(X_exogen = df,forecast_period=1)
    print(prediction)
```

Run

In [16]:

```
full_stats=pd.DataFrame()
ProdCats = ['SUP PREM WHISKEY']
for ProdCat in ProdCats:
    for ChainMaster in ChainMasters:
        printLog("Running ",[ProdCat,ChainMaster])
        ProductsList = getTopProducts(dataRaw, ChainMaster=ChainMaster, ProdCat=ProdCat, topN=TOP_PRODUCTS, timeCol='WeekDate')
        all_stats=runModels(ProductsList,dataRaw,ChainMaster)
        all_stats['Product Category']=ProdCat
        display(all_stats)
        full_stats=full_stats.append(all_stats,ignore_index=True)

printLog("Completed","")
```

```
========================== (2020-08-15 12:56:59.569152)
======== Running  ========
==== SUP PREM WHISKEY ====
============  ============
==========================



================== (2020-08-15 12:56:59.598186)
==== GET DATA ====
========  ========
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1', '2']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2']
% of weeks without a purchase: 1.1904761904761905
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1']
% of weeks without a purchase: 0.0
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
```

```
===================== (2020-08-15 12:57:27.677155)
==== WHITE NOISE ====
=========  ==========
=====================

      lb_stat  lb_pvalue
12  17.529696   0.130735
24  31.092108   0.151145
48  54.922995   0.228882

Is Data White Noise: True
       lb_stat     lb_pvalue
12  115.750529  4.333814e-19
24  214.169023  1.845834e-32
48  308.098428  1.176533e-39

Is Data White Noise: False
       lb_stat     lb_pvalue
12   76.707883  1.745018e-11
24  131.122583  9.711501e-17
48  214.957410  5.416649e-23

Is Data White Noise: False


==================== (2020-08-15 12:57:27.697152)
==== TEST/TRAIN ====
=========  =========
====================

(83, 5) (1, 5)
(83, 5) (1, 5)
(83, 5) (1, 5)


=============== (2020-08-15 12:57:27.703153)
==== NAIVE ====
======  =======
===============

  ID Best Type  Best RMSE All Naives  \
0  0      mean   0.043169         []
1  1      last   0.324455         []
2  2      mean   0.427990         []

                                             All RMSEs  \
```

```
0  [[0.07143315389555474, 0.11332894033533458, 0....
1  [[0.3244545696539494, 0.4963154566538675, 0.34...
2  [[0.6704496090247218, 0.719929333484661, 0.427...

                       All Types                               Best Naive
0  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
1  [[last, seasonal_last, mean]]                   NaiveForecaster(sp=4)
2  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')


===================== (2020-08-15 12:57:28.091184)
==== Multivar P0 ====
=========  ==========
=====================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  3.2min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


========================= (2020-08-15 13:00:38.047215)
==== Multivar P0+Sim ====
===========  ============
=========================


[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  3.1min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name	P0+Sim Best Model RMSE	
0	JACK DANIELS BLK WHSKY 1L		True	mean	0.043169	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.027899	<auto_ts.AutoTimeSeries object at 0x00000228CD...	SARIMAX	0.021390	<aut
1	JACK DANIELS BLK WHSKY 1.75L		False	last	0.324455	NaiveForecaster(sp=4)	auto_SARIMAX	0.207729	<auto_ts.AutoTimeSeries object at 0x00000228CD...	auto_SARIMAX	0.240375	<aut
2	JACK DANIELS BLK WHSKY 750M		False	mean	0.427990	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.278152	<auto_ts.AutoTimeSeries object at 0x00000228CD...	auto_SARIMAX	0.312962	<aut

```
========================== (2020-08-15 13:03:44.109830)
======== Running  ========
==== SUP PREM WHISKEY ====
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-15 13:03:44.127829)
======== GET DATA ========
==== THE BARREL HOUSE ====
==========================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'GENTLEMAN JACK WHSKY 6PK 1L', '2': 'JACK DANIELS BLK WHSKY LSE 50
M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1', '2']
% of weeks without a purchase: 45.23809523809524
resampling to  M



--------------------------------------------------
Product: GENTLEMAN JACK WHSKY 6PK 1L
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2']
% of weeks without a purchase: 32.926829268292686
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY LSE 50M
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1']
% of weeks without a purchase: 10.714285714285714
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: GENTLEMAN JACK WHSKY 6PK 1L
        Product: JACK DANIELS BLK WHSKY LSE 50M
```

```
========================== (2020-08-15 13:04:04.921208)
====== WHITE NOISE =======
==== THE BARREL HOUSE ====
==========================

       lb_stat  lb_pvalue
12   24.932218   0.015147
24   47.266936   0.003107
48  103.010327   0.000007

Is Data White Noise: False
      lb_stat  lb_pvalue
12   8.641692   0.733192
24  19.315056   0.734985
48  47.523727   0.492264

Is Data White Noise: True
      lb_stat  lb_pvalue
12  17.236016   0.140933
24  26.001364   0.353096
48  67.920769   0.030671

Is Data White Noise: True


========================== (2020-08-15 13:04:04.932208)
======= TEST/TRAIN =======
==== THE BARREL HOUSE ====
==========================

(83, 5) (1, 5)
(81, 5) (1, 5)
(83, 5) (1, 5)


========================== (2020-08-15 13:04:04.937208)
========= NAIVE ==========
==== THE BARREL HOUSE ====
==========================

  ID Best Type  Best RMSE All Naives  \
0  0      mean   1.149908         []
1  1      mean   0.611153         []
2  2      mean   0.252773         []

                                           All RMSEs  \
```

```
0  [[1.3378491195367055, 1.5550765479093094, 1.14...
1  [[0.7042605406907361, 0.7703384907350979, 0.61...
2  [[0.4339763671614111, 0.5139045586878941, 0.25...

                    All Types                               Best Naive
0  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
1  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
2  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')


========================== (2020-08-15 13:04:05.143239)
====== Multivar P0 =======
==== THE BARREL HOUSE ====
==========================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  2.9min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


========================== (2020-08-15 13:06:56.390273)
==== Multivar P0+Sim =====
==== THE BARREL HOUSE ====
==========================


[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  2.8min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name	P0+Sim Best Model RMSE	P(
0	JACK DANIELS BLK WHSKY 1L	THE BARREL HOUSE	False	mean	1.149908	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.712256	<auto_ts.AutoTimeSeries object at 0x00000228CC...	ML	0.794521	<auto_
1	GENTLEMAN JACK WHSKY 6PK 1L	THE BARREL HOUSE	True	mean	0.611153	NaiveForecaster(sp=4, strategy='mean')	ML	0.191544	<auto_ts.AutoTimeSeries object at 0x00000228CC...	ML	0.191544	<auto_
2	JACK DANIELS BLK WHSKY LSE 50M	THE BARREL HOUSE	True	mean	0.252773	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.254032	<auto_ts.AutoTimeSeries object at 0x00000228CC...	ML	0.267358	<auto_

```
============================ (2020-08-15 13:09:43.362646)
========= Running  =========
===== SUP PREM WHISKEY =====
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-15 13:09:43.384646)
========= GET DATA =========
==== WESTERN BEV LIQ TX ====
============================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1.75L', '1': 'JACK DANIELS BLK WHSKY  750M', '2': 'JACK DANIELS BLK WHSKY  1
L'}



------------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
------------------------------------------------------
Exogenous Price Columns: ['0', '2', '1']
% of weeks without a purchase: 17.5
resampling to  M



------------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
------------------------------------------------------
Exogenous Price Columns: ['1', '2', '0']
% of weeks without a purchase: 13.414634146341465
resampling to  M



------------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
------------------------------------------------------
Exogenous Price Columns: ['2', '1', '0']
% of weeks without a purchase: 0.0
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS BLK WHSKY  1L
```

```
============================ (2020-08-15 13:10:10.098617)
======= WHITE NOISE ========
==== WESTERN BEV LIQ TX ====
============================

       lb_stat      lb_pvalue
12  230.533574   1.544168e-42
24  440.855949   2.925831e-78
48  689.590939  1.731957e-114

Is Data White Noise: False
       lb_stat     lb_pvalue
12   77.815839  1.075181e-11
24  136.909527  8.579214e-18
48  201.387337  1.088599e-20

Is Data White Noise: False
      lb_stat     lb_pvalue
12  59.487140  2.799051e-08
24  75.433349  3.193517e-07
48  84.194254  9.632108e-04

Is Data White Noise: False


============================ (2020-08-15 13:10:10.113622)
======== TEST/TRAIN ========
==== WESTERN BEV LIQ TX ====
============================

(79, 5) (1, 5)
(81, 5) (1, 5)
(82, 5) (1, 5)


============================ (2020-08-15 13:10:10.119619)
========== NAIVE ===========
==== WESTERN BEV LIQ TX ====
============================

  ID Best Type  Best RMSE All Naives  \
0  0      mean   1.192388         []
1  1      mean   0.706690         []
2  2      mean   0.083096         []

                                             All RMSEs  \
```

```
0  [[1.209792431243374, 1.8312196882995284, 1.192...
1  [[1.1305923379415006, 1.2624589917738127, 0.70...
2  [[0.12236276035592478, 0.0844886828891572, 0.0...

                       All Types                            Best Naive
0  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
1  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
2  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')


============================ (2020-08-15 13:10:10.338648)
======= Multivar P0 ========
==== WESTERN BEV LIQ TX ====
============================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  2.9min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


============================ (2020-08-15 13:13:05.997572)
===== Multivar P0+Sim ======
==== WESTERN BEV LIQ TX ====
============================


[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  3.0min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name	P0+Sim Best Model RMSE	
0	JACK DANIELS BLK WHSKY 1.75L	WESTERN BEV LIQ TX	False	mean	1.192388	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.713414	<auto_ts.AutoTimeSeries object at 0x00000228CD...	SARIMAX	0.651764	<
1	JACK DANIELS BLK WHSKY 750M	WESTERN BEV LIQ TX	False	mean	0.706690	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.578826	<auto_ts.AutoTimeSeries object at 0x00000228CD...	auto_SARIMAX	0.525101	<
2	JACK DANIELS BLK WHSKY 1L	WESTERN BEV LIQ TX	False	mean	0.083096	NaiveForecaster(sp=4, strategy='mean')	ML	0.099956	<auto_ts.AutoTimeSeries object at 0x00000228CA...	ML	0.099927	<

```
========================== (2020-08-15 13:16:05.493945)
======== Running  ========
==== SUP PREM WHISKEY ====
========= SPECS ==========
==========================



================== (2020-08-15 13:16:05.517972)
==== GET DATA ====
===== SPECS ======
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1', '2']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0', '2']
% of weeks without a purchase: 8.333333333333332
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0', '1']
% of weeks without a purchase: 2.380952380952381
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
```

```
==================== (2020-08-15 13:16:27.621236)
==== WHITE NOISE ====
======= SPECS =======
=====================

      lb_stat  lb_pvalue
12  15.190957   0.231159
24  26.538512   0.326419
48  48.667832   0.445959

Is Data White Noise: True
      lb_stat  lb_pvalue
12  29.265420   0.003598
24  41.411630   0.015005
48  54.239869   0.248702

Is Data White Noise: True
      lb_stat  lb_pvalue
12  26.913026   0.007953
24  38.221972   0.032900
48  54.395929   0.244080

Is Data White Noise: True


==================== (2020-08-15 13:16:27.630254)
==== TEST/TRAIN ====
====== SPECS =======
====================

(83, 5) (1, 5)
(83, 5) (1, 5)
(83, 5) (1, 5)


=============== (2020-08-15 13:16:27.635206)
==== NAIVE ====
==== SPECS ====
===============

  ID Best Type  Best RMSE All Naives  \
0  0      mean   0.060885         []
1  1      mean   0.159196         []
2  2      mean   0.276661         []

                                           All RMSEs  \
```

```
0  [[0.10753730489356994, 0.13940924570407018, 0....
1  [[0.1620938777280673, 0.22871564971857916, 0.1...
2  [[0.4671173585435577, 0.5316177371745061, 0.27...

                      All Types                             Best Naive
0  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
1  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')
2  [[last, seasonal_last, mean]]  NaiveForecaster(sp=4, strategy='mean')


===================== (2020-08-15 13:16:27.838210)
==== Multivar P0 ====
======= SPECS =======
=====================


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  2.3min finished
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.


========================= (2020-08-15 13:18:47.134372)
==== Multivar P0+Sim ====
========= SPECS =========
=========================


[Parallel(n_jobs=-1)]: Done   3 out of   3 | elapsed:  3.2min finished
```

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	Naive Best Model	P0 Best Model Name	P0 Best Model RMSE	P0 Best Model	P0+Sim Best Model Name	P0+Sim Best Model RMSE	P0+S
0	JACK DANIELS BLK WHSKY 1L	SPECS	True	mean	0.060885	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.037469	<auto_ts.AutoTimeSeries object at 0x00000228CF...	SARIMAX	0.033760	<auto_ts.A 0x
1	JACK DANIELS BLK WHSKY 1.75L	SPECS	True	mean	0.159196	NaiveForecaster(sp=4, strategy='mean')	SARIMAX	0.119452	<auto_ts.AutoTimeSeries object at 0x00000228CD...	SARIMAX	0.137105	<auto_ts.A 0x
2	JACK DANIELS BLK WHSKY 750M	SPECS	True	mean	0.276661	NaiveForecaster(sp=4, strategy='mean')	auto_SARIMAX	0.237800	<auto_ts.AutoTimeSeries object at 0x00000228CD...	SARIMAX	0.249464	<auto_ts.A 0x

```
=================== (2020-08-15 13:21:59.815404)
==== Completed ====
========  =========
===================
```

Print out

In [15]:

```
full_stats[full_stats.columns.difference(['P0 Best Model','P0+Sim Best Model','Naive Best Model'],sort=False)]
```

Out[15]:

	Product	Chain Master	White Noise	Naive Best Type	Naive Best RMSE	P0 Best Model Name	P0 Best Model RMSE	P0+Sim Best Model Name	P0+Sim Best Model RMSE	Product Category
0	JACK DANIELS BLK WHSKY 1L		True	mean	0.043169	SARIMAX	0.027899	ML	0.030409	SUP PREM WHISKEY
1	JACK DANIELS BLK WHSKY 1.75L		False	last	0.324455	auto_SARIMAX	0.207729	auto_SARIMAX	0.186881	SUP PREM WHISKEY
2	JACK DANIELS BLK WHSKY 750M		False	mean	0.427990	SARIMAX	0.278152	SARIMAX	0.258481	SUP PREM WHISKEY
3	JACK DANIELS BLK WHSKY 1L	THE BARREL HOUSE	False	mean	1.149908	auto_SARIMAX	0.712256	ML	0.798112	SUP PREM WHISKEY
4	GENTLEMAN JACK WHSKY 6PK 1L	THE BARREL HOUSE	True	mean	0.611153	ML	0.191544	ML	0.191544	SUP PREM WHISKEY
5	JACK DANIELS BLK WHSKY LSE 50M	THE BARREL HOUSE	True	mean	0.252773	auto_SARIMAX	0.254032	auto_SARIMAX	0.261385	SUP PREM WHISKEY
6	JACK DANIELS BLK WHSKY 1.75L	WESTERN BEV LIQ TX	False	mean	1.192388	auto_SARIMAX	0.713414	Prophet	0.739690	SUP PREM WHISKEY
7	JACK DANIELS BLK WHSKY 750M	WESTERN BEV LIQ TX	False	mean	0.706690	SARIMAX	0.578826	auto_SARIMAX	0.514482	SUP PREM WHISKEY
8	JACK DANIELS BLK WHSKY 1L	WESTERN BEV LIQ TX	False	mean	0.083096	ML	0.099956	auto_SARIMAX	0.089549	SUP PREM WHISKEY
9	JACK DANIELS BLK WHSKY 1L	SPECS	True	mean	0.060885	SARIMAX	0.037469	ML	0.039430	SUP PREM WHISKEY
10	JACK DANIELS BLK WHSKY 1.75L	SPECS	True	mean	0.159196	SARIMAX	0.119452	SARIMAX	0.137262	SUP PREM WHISKEY
11	JACK DANIELS BLK WHSKY 750M	SPECS	True	mean	0.276661	auto_SARIMAX	0.237800	SARIMAX	0.233663	SUP PREM WHISKEY

Saving

In [16]:

```
#full_stats.to_pickle('all_Models_stats.pkl')
```

Optimizer

Functions

Optimizer Functions

In [17]:

```
def complex_objective(x: List
                      , ts_index_name: str
                      , ts_index: List
                      , all_models: List
                      , all_data: List
                      , mask: Optional[List[bool]] = None
                      , verbose: int = 0
                      , return_individual: bool = False
                      , logT = False
                      , P0_only = False
                      #argument for P0 only
                      ):
    """
    :param x A list of product pricing for which the revenue has to be computed
    :type x List
    :param mask: If the customer is not going to purchase a product in a period, we can choose to omit it from the revenue calculat
ion in the optimizer.
                 Default = None (considers all products in revenue calculation)
    :type mask  Optional[List[bool]]

    :param ts_index The index to use for the test data. This is needed for some models (such as ML) that use this to create feature
s
    :type ts_index List

    :param return_individual If True, this returns the individual revenue values as well
                             Used mainly when this function is called standalone. Set of False for optimization
    :type return_individual bool

    :param verbose Level of verbosity (Default: 0). This is set to 1 or 2 (mainly for debug purposes)
    :type verbose int
    """
    if verbose >0: print ("### Prediction Function ###")
    # Create test data from input
    index = [str(i) for i in np.arange(len(x))]
    x_df = pd.DataFrame(x, index = index)
    x_df = x_df.T

    # Set index (important for some models)
    x_df.index = ts_index
    x_df.index.name = ts_index_name

    # If mask is not provided, use all
    if mask is None:
        mask = [False for item in x]
```

```python
    if verbose >= 2:
        print(x_df.info())
        print(x_df.columns)

    total_revenue = 0
    revenue = []

    for i in np.arange(len(all_data)):
        if verbose >= 1:
            print("\n" + "-"*50)
            print(f"Product Index: {i}")

        if not mask[i]:
            if P0_only: columns = [all_data[i].columns[-(TOP_SIMILAR+1)]]
            else: columns = all_data[i].columns[-(TOP_SIMILAR+1):].values #columns[-(TOP_SIMILAR+2)] for the P0 only type
            if verbose >= 2:
                print(f"All Columns in Test Data: {columns}")
                print('i:',i)
                print(x_df[columns])
                print("-----------------------------------")

            test_data = x_df[columns]
            prediction = all_models[i].predict(X_exogen = test_data,forecast_period=1) #change this back when Nikhil fixes the auto
TS

            if verbose >= 2: print(f"Prediction Type: {type(prediction)}")
            if verbose >= 1: print(f"Demand Prediction (transformed): {prediction}")

            # If model was created with log transformation
            if logT:
                prediction = 10**prediction
                if verbose >= 1:
                    print("\nDemand Prediction (Original)")
                    print(prediction)

            product_revenue = prediction * x[i]

            # TODO: Clamping - Fix later (this gives an error with pandas. We need to pluck it out as a value)
            # product_revenue = max(product_revenue, 0)  # Clamp at min value of 0 for predictions that are negative

            if verbose >= 1: print(f"Product Revenue: ${round(product_revenue)}")

            if isinstance(product_revenue, pd.Series):
                product_revenue = product_revenue.iloc[0]
            revenue.append(product_revenue)

            # total_revenue = total_revenue + product_revenue
        else:
```

```
            if verbose >= 1: print("This product's revenue was not included since it was not ordered by the customer in this perio
d.")
            product_revenue = 0
            revenue.append(product_revenue)

        if verbose >= 1: print("-"*50 + "\n")

    total_revenue = sum(revenue)

    if verbose >= 1:
        print("\n\n" + "="*50)
        print(f"Total Revenue: ${round(total_revenue)}")
        print("="*50 + "\n\n")
        print ("### Prediction Function END ###")
    if return_individual is True: return -total_revenue, revenue

    return -total_revenue
```

Core Functions

In [18]:

```
def opt_get_mask(all_data,all_test):
    # Did the customer actually want to but products in that period?
    # Only include the revenue in the objective if they actually ordered it
    # This model is not trying to predict if they would purchase a product when they were not going to purchase it earlier.
    # That requires a lot of human psychology and may not be captured in the model

    INCLUDE_MASKING = True

    mask: List[bool] = []
    for index in np.arange(len(all_data)):
        if INCLUDE_MASKING:
            if all_test[index].iloc[0]['9L Cases'] == 0:
                mask.append(True)
            else:
                mask.append(False)
        else:
            mask.append(False)

    print(f"Mask: {mask}")
    return(mask)

def opt_get_space(all_data,MARGIN=0.0):
    MARGIN = 0.0 # How much to go over or under the min and max price respectively during the search for optimial revenue
    space = []

    for index in np.arange(len(all_data)):
        #min_val = all_data[index][str(index)].min()
        min_val = np.percentile(all_data[index][str(index)], 10)
        #max_val = all_data[index][str(index)].max()
        max_val = np.percentile(all_data[index][str(index)], 90)
        min_limit = min_val*(1-MARGIN)
        max_limit = max_val*(1+MARGIN)
        space.append(Real(low=min_limit, high=max_limit, prior='uniform'))

    return(space)

def opt_get_func(all_data,all_models,complex_objective,test_index_name,test_index,mask,verbose=0,P0_only=False):
    # create a new function with mask
    masked_complex_objective = partial(complex_objective, ts_index_name=test_index_name, ts_index=test_index, mask=mask, logT=LOG_T
RANSFORM,verbose=verbose
                                       ,all_models=all_models,all_data=all_data,P0_only=P0_only)
    if P0_only:
        print(f"Revenue P0: ${-round(complex_objective([266.51, 195.06, 205.3], ts_index_name=test_index_name, ts_index=test_index,
mask=mask,logT=LOG_TRANSFORM,verbose=verbose,all_models=all_models,all_data=all_data,P0_only=True))}")
    else:
```

```
        print(f"Revenue without masking: ${-round(complex_objective([266.51, 195.06, 205.3], ts_index_name=test_index_name, ts_inde
x=test_index, logT=LOG_TRANSFORM,verbose=verbose,all_models=all_models,all_data=all_data))}")
        print(f"Revenue with masking: ${-round(masked_complex_objective([266.51, 195.06, 205.3],verbose=verbose,all_models=all_mode
ls,all_data=all_data))}")
    return(masked_complex_objective)

def opt_get_data(all_data,all_test_non_transformed):
    total_test_data_revenue = 0
    for index in np.arange(len(all_data)):
        product_price = all_test_non_transformed[index].iloc[0][str(index)]
        product_demand = all_test_non_transformed[index].iloc[0]['9L Cases']
        product_revenue = product_price * product_demand
        print(f"Product {index} Price 9L Case: ${round(product_price,2)} Revenue: ${round(product_revenue)}")
        total_test_data_revenue = total_test_data_revenue + product_revenue

    print(f"Total Revenue: ${round(total_test_data_revenue)}")
    return(total_test_data_revenue)

def opt_naive(all_models,all_test_non_transformed):
    #uses test price and predict demand based on naive model
    product_price=[]
    product_demand=[]
    product_revenue=[]
    for index in np.arange(len(all_models)):
        product_price.append(all_test_non_transformed[index].iloc[0][str(index)])
        product_demand.append(all_models[index].predict([0]).tolist()[0])
        product_revenue.append(product_price[index] * product_demand[index])
    total_revenue = sum(product_revenue)
    return(product_price,product_demand,product_revenue,total_revenue)

def opt_get_chart(all_data,all_models,space,ChainMaster,ProdCat,test_index,test_index_name,verbose=1,STEPS=5,displayPlots=True,save
Path = '3d_charts/'):
    math.ceil(space[0].low)
    math.floor(space[0].high)
    xs = np.arange(math.ceil(space[0].low), math.floor(space[0].high), step=5)
    ys = np.arange(math.ceil(space[1].low), math.floor(space[1].high), step=5)

    allp = [np.arange(math.ceil(space[i].low), math.floor(space[i].high), step=STEPS) for i in np.arange(len(all_data))]

    if verbose >= 1:
        print("-"*100)
        print(f"Price intervals for product 0: {allp[0]}")
        print(f"Price intervals for product 1: {allp[1]}")
        print(f"Price intervals for product 2: {allp[2]}")
        print("-"*100, "\n")
    filenames=[]
    for i in np.arange(len(all_data)):
        print("\n\n")
```

```
        mask_plot = [False if i == j else True for j in np.arange(len(all_data))]
        if verbose >= 1:
            print(f"Product {i} --> Mask: {mask_plot}")

        columns = all_data[i].columns[-(TOP_SIMILAR+1):].values
        if verbose >= 1:
            print(f"Products used in Model: {columns}")

        masked_complex_objective_plot = partial(complex_objective, ts_index_name=test_index_name, ts_index=test_index, mask=mask_pl
ot, logT=LOG_TRANSFORM, verbose=0
                                                ,all_models=all_models,all_data=all_data)

        finalx = []
        finaly = []
        finalrev = []

        xs = allp[int(columns[0])]  # Main Product Price is in xs
        ys = allp[int(columns[1])]  # Exogenous Product Price in in ys

        if verbose >= 1:
            print(f"Price intervals used for X-axis (product {int(columns[0])}): {xs}")
            print(f"Price intervals used for Y-axis (product {int(columns[1])}): {ys}")

        for x, y in itertools.product(xs, ys):
            price_list = [0, 0, 0]

            # Fix price for product 0
            if int(columns[0]) == 0:  # If the main product is product 0
                price_list[0] = x
            elif int(columns[1]) == 0: # If exogenous product is product 0
                price_list[0] = y
            else:
                price_list[0] = 0

            # Fix price for product 1
            if int(columns[0]) == 1:  # If the main product is product 1
                price_list[1] = x
            elif int(columns[1]) == 1: # If exogenous product is product 1
                price_list[1] = y
            else:
                price_list[1] = 0

            # Fix price for product 2
            if int(columns[0]) == 2:  # If the main product is product 2
                price_list[2] = x
            elif int(columns[1]) == 2: # If exogenous product is product 2
                price_list[2] = y
            else:
```

```
            price_list[2] = 0

        rev = -masked_complex_objective_plot(price_list)
        finalx.append(x)
        finaly.append(y)
        finalrev.append(rev)

    fig = surface3DChart(
        x=finalx, y=finaly, z=finalrev,
        title= 'Product ' + columns[0] + ' Revenue',
        xTitle= 'Product ' + columns[0] + ' Price',
        yTitle= 'Product ' + columns[1] + ' Price',
        width=1200,
        height=800
        )

    filename = "".join(ChainMaster.split()) + "_" + "".join(ProdCat.split()) + "_Top" + str(TOP_PRODUCTS) + "_Sim" + str(TOP_SI
MILAR) + \
        "_Log" + str(LOG_TRANSFORM) + "_Add" + str(ZERO_ADDER) + \
        "_Prod" + str(i) + "_Resample" + str(RESAMPLE_FREQ) + "_f" + str(FORECAST_PERIOD) + "_s" + str(SEASONAL_PERIOD) + ".htm
l"
    filenameFull = os.path.join(savePath,filename)
    if verbose >=1: print(filenameFull)
    filenames.append(filenameFull)
    py.plot(fig, filename = filenameFull,auto_open=displayPlots)
return(filenames)
```

Call Function

In [24]:

```
def runOptimizer(ProductsList,dataRaw,ChainMaster,modelsStats,verbose=0):
    opt_stats = pd.DataFrame()
    numProducts = len(ProductsList)
    opt_stats['Chain Master'] = [ChainMaster] * numProducts
    opt_stats['Product'] = ProductsList


    printLog("GET DATA",ChainMaster)
    all_data,all_data_non_transformed,colExog,colEnc,colDec = modelsLoadData(ProductsList,dataRaw,ChainMaster)

    printLog("TEST/TRAIN",ChainMaster)
    all_train, all_test, all_train_non_transformed, all_test_non_transformed = ModelsTestTrain(all_data,all_data_non_transformed)
    opt_stats['Actual Demand'] = [all_test_non_transformed[x]['9L Cases'].values[0] for x in np.arange(3)]
    opt_stats['Actual Price'] = [all_test_non_transformed[x].iloc[0][str(x)] for x in np.arange(3)]
    opt_stats['Actual Revenue'] =  [opt_stats['Actual Demand'][x] * opt_stats['Actual Price'][x]  for x in np.arange(numProducts)]
    opt_stats['Actual Chain Master Revenue'] =  [sum(opt_stats['Actual Revenue'])] *numProducts

    printLog("NAIVE FORECAST",ChainMaster)
    all_models = modelsStats['Naive Best Model']
    naive_price, naive_demand, naive_revenue ,naive_total_revenue = opt_naive(all_models,all_test_non_transformed) #uses test price
and predict demand based on naive
    opt_stats['Naive Prices'] = naive_price
    opt_stats['Naive Demand'] = naive_demand
    opt_stats['Naive Revenue'] = naive_revenue
    opt_stats['Naive Chain Master Revenue'] = [naive_total_revenue] * numProducts

    printLog("MASK",ChainMaster)
    mask = opt_get_mask(all_data,all_test)
    opt_stats['mask'] = mask

    printLog("SPACE",ChainMaster)
    space = opt_get_space(all_data)
    opt_stats['space'] = space

    printLog("Test Index",ChainMaster)
    test_index_name = 'WeekDate'
    test_index = all_test_non_transformed[0][test_index_name].values
    opt_stats['test_index'] = [test_index] * numProducts# for i in ProductsList]

    #############
    ## P0 Only ##
    if True:
        printLog("GET FUNCTION P0",ChainMaster)
        all_models = modelsStats['P0 Best Model']
        masked_complex_objective = opt_get_func(all_data,all_models,complex_objective,test_index_name,test_index,mask=mask,verbose=
```

```
verbose,P0_only=True)
        opt_stats['masked_complex_objective'] = masked_complex_objective

        printLog("OPTIMIZING P0",ChainMaster)
        res = gp_minimize(masked_complex_objective,
                          space,
                          acq_func="EI",
                          n_calls=200,
                          n_random_starts=20,
                          random_state=42)
        opt_stats['res'] = [res] * numProducts # for i in ProductsList]

        ## GET OUTPUT DATA ##
        printLog("OUTPUT P0",ChainMaster)
        opt_stats['P0 Optimal Price'] = [round(price, 2) for price in res.x]
        opt_stats['P0 Chain Master Revenue'] = round(-res.fun)

        _,all_revenues =  masked_complex_objective(res.x, return_individual=True)
        opt_stats['P0 Demand'] = (np.array(all_revenues) / np.array(opt_stats['P0 Optimal Price'])).tolist()
        opt_stats['P0 Revenue'] = all_revenues

        total_test_data_revenue = opt_get_data(all_data,all_test_non_transformed)
        opt_stats['total_test_data_revenue_P0'] = total_test_data_revenue

    ############
    ## P0+Sim ##
    printLog("GET FUNCTION P0+Sim",ChainMaster)
    all_models = modelsStats['P0+Sim Best Model']
    masked_complex_objective = opt_get_func(all_data,all_models,complex_objective,test_index_name,test_index,mask,verbose=verbose,P
0_only=False)
    opt_stats['masked_complex_objective'] = masked_complex_objective

    printLog("OPTIMIZING P0+Sim",ChainMaster)
    res = gp_minimize(masked_complex_objective,
                      space,
                      acq_func="EI",
                      n_calls=200,
                      n_random_starts=20,
                      random_state=42
                     )
    opt_stats['res'] = [res] * numProducts # for i in ProductsList]

    ## GET OUTPUT DATA ##
    printLog("OUTPUT P0+Sim",ChainMaster)
    opt_stats['P0+Sim Optimal Price'] = [round(price, 2) for price in res.x]
    opt_stats['P0+Sim Chain Master Revenue'] = round(-res.fun)

    _,all_revenues =  masked_complex_objective(res.x, return_individual=True)
```

```
    opt_stats['P0+Sim Demand'] = (np.array(all_revenues) / np.array(opt_stats['P0+Sim Optimal Price'])).tolist()
    opt_stats['P0+Sim Revenue'] = all_revenues

    total_test_data_revenue = opt_get_data(all_data,all_test_non_transformed)
    opt_stats['total_test_data_revenue_P0+Sim'] = total_test_data_revenue

    ############
    # 3D Charts ##
    if False:
        printLog("3D CHARTS",ChainMaster)
        filenames = opt_get_chart(all_data,all_models,space,ChainMaster,ProdCat,test_index,test_index_name,verbose=1,STEPS=5,displa
yPlots=False)
        opt_stats['3d_chart_filenames']  = filenames

    printLog("COMPLETED",ChainMaster)

    return(opt_stats)
```

Loop

In [25]:

```
#reading models data
#full_stats = pd.read_pickle('all_Models_stats.pkl')
#check mask.. change the iteration to 10 random and 20 full
```

In [26]:

```
ChainMasters =  [''] +  dataRaw['Chain Master'].unique().tolist()
ProdCats = dataRaw['Category (CatMan)'].unique().tolist()
display(ChainMasters,ProdCats)
```

```
['', 'THE BARREL HOUSE', 'WESTERN BEV LIQ TX', 'SPECS']

['ECONOMY VODKA', 'SUP PREM WHISKEY']
```

Testing Models

In [27]:

```
## testing Models Prediction
if False:
    ChainMaster = ChainMasters[2]#Western
    ProdCat = 'SUP PREM WHISKEY'
    modelsStats = full_stats[(full_stats['Chain Master']==ChainMaster) & (full_stats['Product Category']==ProdCat)].reset_index()
    display(modelsStats)
    #display(modelsStats)
    model = modelsStats['P0 Best Model'][1]
    #df=pd.DataFrame({'WeekDate': [pd.to_datetime('2019-12-31')],'0':[266.51],'1':[195.06],'2':[195.06]})
    df=pd.DataFrame({'WeekDate': [pd.to_datetime('2019-12-31')],'1':[266.51]})
    prediction=model.predict(X_exogen = df,forecast_period=1)
    print(prediction)
```

In [28]:

```
full_opt_stats=pd.DataFrame()
ProdCats = ['SUP PREM WHISKEY']
for ProdCat in ProdCats:
    for ChainMaster in ChainMasters:
        modelsStats = full_stats[(full_stats['Chain Master']==ChainMaster) & (full_stats['Product Category']==ProdCat)].reset_index
()

        printLog("Get Top Similar Products",[ProdCat,ChainMaster])
        ProductsList = getTopProducts(dataRaw, ChainMaster=ChainMaster, ProdCat=ProdCat, topN=TOP_PRODUCTS, timeCol='WeekDate')

        printLog("Running Optimizer",[ProdCat,ChainMaster])
        opt_stats=runOptimizer(ProductsList,dataRaw,ChainMaster,modelsStats,verbose=0)

        #display(opt_stats)
        full_opt_stats=full_opt_stats.append(opt_stats,ignore_index=True)

printLog("Completed","")
```

```
================================== (2020-08-09 11:00:12.928363)
==== Get Top Similar Products ====
======== SUP PREM WHISKEY ========
================  ================
==================================



=========================== (2020-08-09 11:00:12.957287)
==== Running Optimizer ====
==== SUP PREM WHISKEY =====
============  ============
===========================



================== (2020-08-09 11:00:12.958285)
==== GET DATA ====
========  ========
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0']
% of weeks without a purchase: 1.1904761904761905
resampling to  M



--------------------------------------------------
```

```
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0']
% of weeks without a purchase: 0.0
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M


==================== (2020-08-09 11:00:35.995890)
==== TEST/TRAIN ====
=========  =========
====================

(83, 4) (1, 4)
(83, 4) (1, 4)
(83, 4) (1, 4)


======================== (2020-08-09 11:00:35.999879)
==== NAIVE FORECAST ====
===========  ===========
========================




============== (2020-08-09 11:00:36.009851)
==== MASK ====
======  ======
==============

Mask: [False, False, False]


=============== (2020-08-09 11:00:36.010877)
==== SPACE ====
======  =======
===============




==================== (2020-08-09 11:00:36.014839)
==== Test Index ====
=========  =========
====================
```

```
========================= (2020-08-09 11:00:36.014839)
==== GET FUNCTION P0 ====
===========  ============
=========================

Revenue P0: $272192.0


======================= (2020-08-09 11:00:36.051739)
==== OPTIMIZING P0 ====
==========  ===========
=======================



=================== (2020-08-09 11:03:20.233514)
==== OUTPUT P0 ====
========  =========
===================

Product 0 Price 9L Case: $229.81 Revenue: $135402.0
Product 1 Price 9L Case: $185.65 Revenue: $72331.0
Product 2 Price 9L Case: $222.36 Revenue: $50031.0
Total Revenue: $257765.0


============================= (2020-08-09 11:03:20.292356)
==== GET FUNCTION P0+Sim ====
=============  ==============
=============================

Revenue without masking: $214111.0
Revenue with masking: $214111.0


=========================== (2020-08-09 11:03:20.428991)
==== OPTIMIZING P0+Sim ====
============  =============
===========================



======================= (2020-08-09 11:06:09.831987)
==== OUTPUT P0+Sim ====
==========  ===========
=======================
```

```
Product 0 Price 9L Case: $229.81 Revenue: $135402.0
Product 1 Price 9L Case: $185.65 Revenue: $72331.0
Product 2 Price 9L Case: $222.36 Revenue: $50031.0
Total Revenue: $257765.0


=================== (2020-08-09 11:06:09.907785)
==== COMPLETED ====
========  =========
===================



================================== (2020-08-09 11:06:09.923743)
==== Get Top Similar Products ====
======== SUP PREM WHISKEY ========
======== THE BARREL HOUSE ========
==================================



=========================== (2020-08-09 11:06:09.946681)
==== Running Optimizer ====
==== SUP PREM WHISKEY =====
==== THE BARREL HOUSE =====
===========================



========================== (2020-08-09 11:06:09.949673)
======== GET DATA ========
==== THE BARREL HOUSE ====
==========================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'GENTLEMAN JACK WHSKY 6PK 1L', '2': 'JACK DANIELS BLK WHSKY LSE 50
M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1']
% of weeks without a purchase: 45.23809523809524
resampling to  M
```

```
--------------------------------------------------
Product: GENTLEMAN JACK WHSKY 6PK 1L
--------------------------------------------------
Exogenous Price Columns: ['1', '0']
% of weeks without a purchase: 32.926829268292686
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY LSE 50M
--------------------------------------------------
Exogenous Price Columns: ['2', '0']
% of weeks without a purchase: 10.714285714285714
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: GENTLEMAN JACK WHSKY 6PK 1L
        Product: JACK DANIELS BLK WHSKY LSE 50M


========================== (2020-08-09 11:06:34.166684)
======= TEST/TRAIN =======
==== THE BARREL HOUSE ====
==========================

(83, 4) (1, 4)
(81, 4) (1, 4)
(83, 4) (1, 4)


========================== (2020-08-09 11:06:34.172670)
===== NAIVE FORECAST =====
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-09 11:06:34.186660)
========== MASK ==========
==== THE BARREL HOUSE ====
==========================

Mask: [False, False, False]


========================== (2020-08-09 11:06:34.188625)
========= SPACE ==========
```

```
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-09 11:06:34.194638)
======= Test Index =======
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-09 11:06:34.194638)
==== GET FUNCTION P0 =====
==== THE BARREL HOUSE ====
==========================

Revenue P0: $557.0


========================== (2020-08-09 11:06:34.256446)
===== OPTIMIZING P0 ======
==== THE BARREL HOUSE ====
==========================



========================== (2020-08-09 11:09:35.144127)
======= OUTPUT P0 ========
==== THE BARREL HOUSE ====
==========================

Product 0 Price 9L Case: $239.01 Revenue: $636.0
Product 1 Price 9L Case: $286.87 Revenue: $1345.0
Product 2 Price 9L Case: $268.66 Revenue: $360.0
Total Revenue: $2341.0


============================= (2020-08-09 11:09:35.223887)
==== GET FUNCTION P0+Sim ====
===== THE BARREL HOUSE ======
=============================

Revenue without masking: $867.0
Revenue with masking: $867.0


=========================== (2020-08-09 11:09:35.397425)
```

```
==== OPTIMIZING P0+Sim ====
==== THE BARREL HOUSE =====
===========================



========================== (2020-08-09 11:12:14.024961)
===== OUTPUT P0+Sim ======
==== THE BARREL HOUSE ====
==========================

Product 0 Price 9L Case: $239.01 Revenue: $636.0
Product 1 Price 9L Case: $286.87 Revenue: $1345.0
Product 2 Price 9L Case: $268.66 Revenue: $360.0
Total Revenue: $2341.0


========================== (2020-08-09 11:12:14.113726)
======= COMPLETED ========
==== THE BARREL HOUSE ====
==========================



================================== (2020-08-09 11:12:14.117714)
==== Get Top Similar Products ====
======== SUP PREM WHISKEY ========
======= WESTERN BEV LIQ TX =======
==================================



============================ (2020-08-09 11:12:14.137660)
==== Running Optimizer =====
===== SUP PREM WHISKEY =====
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-09 11:12:14.140653)
========= GET DATA =========
==== WESTERN BEV LIQ TX ====
============================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1.75L', '1': 'JACK DANIELS BLK WHSKY  750M', '2': 'JACK DANIELS BLK WHSKY  1
L'}
```

```
--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['0', '2']
% of weeks without a purchase: 17.5
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['1', '2']
% of weeks without a purchase: 13.414634146341465
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['2', '1']
% of weeks without a purchase: 0.0
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M
        Product: JACK DANIELS BLK WHSKY  1L


============================ (2020-08-09 11:12:37.212821)
======== TEST/TRAIN ========
==== WESTERN BEV LIQ TX ====
============================

(79, 4) (1, 4)
(81, 4) (1, 4)
(82, 4) (1, 4)


============================ (2020-08-09 11:12:37.218802)
====== NAIVE FORECAST ======
==== WESTERN BEV LIQ TX ====
============================
```

```
============================ (2020-08-09 11:12:37.231768)
=========== MASK ===========
==== WESTERN BEV LIQ TX ====
============================

Mask: [False, False, False]


============================ (2020-08-09 11:12:37.233762)
========== SPACE ===========
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-09 11:12:37.238749)
======== Test Index ========
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-09 11:12:37.240744)
===== GET FUNCTION P0 ======
==== WESTERN BEV LIQ TX ====
============================

Revenue P0: $60042276.0


============================ (2020-08-09 11:12:37.327513)
====== OPTIMIZING P0 =======
==== WESTERN BEV LIQ TX ====
============================



============================ (2020-08-09 11:15:36.315638)
======== OUTPUT P0 =========
==== WESTERN BEV LIQ TX ====
============================

Product 0 Price 9L Case: $185.59 Revenue: $39085.0
Product 1 Price 9L Case: $222.36 Revenue: $34466.0
Product 2 Price 9L Case: $230.79 Revenue: $25476.0
Total Revenue: $99027.0
```

```
============================= (2020-08-09 11:15:36.380463)
==== GET FUNCTION P0+Sim ====
==== WESTERN BEV LIQ TX =====
=============================

Building Forecast dataframe. Forecast Period = 1
Revenue without masking: $375471.0
Building Forecast dataframe. Forecast Period = 1
Revenue with masking: $375471.0


============================ (2020-08-09 11:15:39.734542)
==== OPTIMIZING P0+Sim =====
==== WESTERN BEV LIQ TX ====
============================

Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
```

```
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1
Building Forecast dataframe. Forecast Period = 1


============================ (2020-08-09 11:24:14.081043)
====== OUTPUT P0+Sim =======
==== WESTERN BEV LIQ TX ====
============================

Building Forecast dataframe. Forecast Period = 1
Product 0 Price 9L Case: $185.59 Revenue: $39085.0
Product 1 Price 9L Case: $222.36 Revenue: $34466.0
Product 2 Price 9L Case: $230.79 Revenue: $25476.0
Total Revenue: $99027.0


============================ (2020-08-09 11:24:15.757973)
======== COMPLETED =========
==== WESTERN BEV LIQ TX ====
============================




================================== (2020-08-09 11:24:15.762958)
==== Get Top Similar Products ====
```

```
======== SUP PREM WHISKEY ========
============ SPECS =============
==================================



========================== (2020-08-09 11:24:15.788889)
==== Running Optimizer ====
==== SUP PREM WHISKEY =====
========== SPECS ==========
===========================



================== (2020-08-09 11:24:15.792878)
==== GET DATA ====
===== SPECS ======
==================

resampling to  M
Decoder: {'0': 'JACK DANIELS BLK WHSKY  1L', '1': 'JACK DANIELS BLK WHSKY  1.75L', '2': 'JACK DANIELS BLK WHSKY  750
M'}



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1L
--------------------------------------------------
Exogenous Price Columns: ['0', '1']
% of weeks without a purchase: 0.0
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  1.75L
--------------------------------------------------
Exogenous Price Columns: ['1', '0']
% of weeks without a purchase: 8.333333333333332
resampling to  M



--------------------------------------------------
Product: JACK DANIELS BLK WHSKY  750M
--------------------------------------------------
Exogenous Price Columns: ['2', '0']
% of weeks without a purchase: 2.380952380952381
```

```
Log Transforming
        Product: JACK DANIELS BLK WHSKY  1L
        Product: JACK DANIELS BLK WHSKY  1.75L
        Product: JACK DANIELS BLK WHSKY  750M


==================== (2020-08-09 11:24:42.744708)
==== TEST/TRAIN ====
====== SPECS =======
====================

(83, 4) (1, 4)
(83, 4) (1, 4)
(83, 4) (1, 4)


======================== (2020-08-09 11:24:42.749695)
==== NAIVE FORECAST ====
======== SPECS =========
========================




=============== (2020-08-09 11:24:42.758673)
==== MASK =====
==== SPECS ====
===============

Mask: [False, False, False]


=============== (2020-08-09 11:24:42.759669)
==== SPACE ====
==== SPECS ====
===============




==================== (2020-08-09 11:24:42.764657)
==== Test Index ====
====== SPECS =======
====================




========================= (2020-08-09 11:24:42.765654)
==== GET FUNCTION P0 ====
========= SPECS =========
```

```
=========================

Revenue P0: $189945.0


======================= (2020-08-09 11:24:42.822501)
==== OPTIMIZING P0 ====
======== SPECS ========
=======================



=================== (2020-08-09 11:28:18.780189)
==== OUTPUT P0 ====
====== SPECS ======
===================

Product 0 Price 9L Case: $229.53 Revenue: $109290.0
Product 1 Price 9L Case: $185.59 Revenue: $32788.0
Product 2 Price 9L Case: $222.36 Revenue: $15343.0
Total Revenue: $157421.0


============================= (2020-08-09 11:28:18.857493)
==== GET FUNCTION P0+Sim ====
=========== SPECS ===========
=============================

Revenue without masking: $90148.0
Revenue with masking: $90148.0


=========================== (2020-08-09 11:28:19.161198)
==== OPTIMIZING P0+Sim ====
========== SPECS ==========
===========================



======================= (2020-08-09 11:31:50.168376)
==== OUTPUT P0+Sim ====
======== SPECS ========
=======================

Product 0 Price 9L Case: $229.53 Revenue: $109290.0
Product 1 Price 9L Case: $185.59 Revenue: $32788.0
Product 2 Price 9L Case: $222.36 Revenue: $15343.0
Total Revenue: $157421.0
```

```
=================== (2020-08-09 11:31:50.267139)
==== COMPLETED ====
====== SPECS ======
===================
```

```
=================== (2020-08-09 11:31:50.271103)
==== Completed ====
========  =========
===================
```

Print out

In [29]:

```
full_opt_stats[['Chain Master','Product'
                ,'Actual Price','Actual Demand','Actual Revenue','Actual Chain Master Revenue'
                ,'Naive Prices','Naive Demand','Naive Revenue','Naive Chain Master Revenue'
                ,'P0 Optimal Price','P0 Demand','P0 Revenue','P0 Chain Master Revenue'
                ,'P0+Sim Optimal Price','P0+Sim Demand','P0+Sim Revenue','P0+Sim Chain Master Revenue'
                ]]
```

Out[29]:

	Chain Master	Product	Actual Price	Actual Demand	Actual Revenue	Actual Chain Master Revenue	Naive Prices	Naive Demand	Naive Revenue	Naive Chain Master Revenue	P0 Optimal Price	P0 Demand
0		JACK DANIELS BLK WHSKY 1L	229.811232	589.19	135402.48	257764.62	229.811232	292.610723	67245.230832	90321.715676	227.71	296.250933
1		JACK DANIELS BLK WHSKY 1.75L	185.650112	389.61	72331.14	257764.62	185.650112	40.950000	7602.372072	90321.715676	184.44	184.916178
2		JACK DANIELS BLK WHSKY 750M	222.360000	225.00	50031.00	257764.62	222.360000	69.590361	15474.112771	90321.715676	223.78	169.864344
3	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 1L	239.007519	2.66	635.76	2341.20	239.007519	9.200506	2198.990116	2900.866248	222.56	225.475606
4	THE BARREL HOUSE	GENTLEMAN JACK WHSKY 6PK 1L	286.874200	4.69	1345.44	2341.20	286.874200	1.169049	335.370107	2900.866248	295.95	1.484805
5	THE BARREL HOUSE	JACK DANIELS BLK WHSKY LSE 50M	268.656716	1.34	360.00	2341.20	268.656716	1.364217	366.506024	2900.866248	267.99	1.272531
6	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1.75L	185.589744	210.60	39085.20	99027.48	185.589744	110.560063	20518.813797	44989.509807	191.28	166.023589
7	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 750M	222.360000	155.00	34465.80	99027.48	222.360000	41.041148	9125.909702	44989.509807	217.61	165.010479
8	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1L	230.786122	110.39	25476.48	99027.48	230.786122	66.489207	15344.786307	44989.509807	224.24	58.682795
9	SPECS	JACK DANIELS BLK WHSKY 1L	229.533835	476.14	109290.24	157421.22	229.533835	217.722084	49974.584892	63726.848507	226.50	222.317808

	Chain Master	Product	Actual Price	Actual Demand	Actual Revenue	Actual Chain Master Revenue	Naive Prices	Naive Demand	Naive Revenue	Naive Chain Master Revenue	P0 Optimal Price	P0 Demand
10	SPECS	JACK DANIELS BLK WHSKY 1.75L	185.589744	176.67	32788.14	157421.22	185.589744	44.962771	8344.629157	63726.848507	184.04	49.283717
11	SPECS	JACK DANIELS BLK WHSKY 750M	222.360000	69.00	15342.84	157421.22	222.360000	24.319277	5407.634458	63726.848507	205.54	112.562539

In [30]:

```
full_opt_stats[['Chain Master','Product'
                ,'Actual Price','Actual Demand','Actual Revenue','Actual Chain Master Revenue'
                ,'Naive Prices','Naive Demand','Naive Revenue','Naive Chain Master Revenue'
                ]]
```

Out[30]:

	Chain Master	Product	Actual Price	Actual Demand	Actual Revenue	Actual Chain Master Revenue	Naive Prices	Naive Demand	Naive Revenue	Naive Chain Master Revenue
0		JACK DANIELS BLK WHSKY 1L	229.811232	589.19	135402.48	257764.62	229.811232	292.610723	67245.230832	90321.715676
1		JACK DANIELS BLK WHSKY 1.75L	185.650112	389.61	72331.14	257764.62	185.650112	40.950000	7602.372072	90321.715676
2		JACK DANIELS BLK WHSKY 750M	222.360000	225.00	50031.00	257764.62	222.360000	69.590361	15474.112771	90321.715676
3	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 1L	239.007519	2.66	635.76	2341.20	239.007519	9.200506	2198.990116	2900.866248
4	THE BARREL HOUSE	GENTLEMAN JACK WHSKY 6PK 1L	286.874200	4.69	1345.44	2341.20	286.874200	1.169049	335.370107	2900.866248
5	THE BARREL HOUSE	JACK DANIELS BLK WHSKY LSE 50M	268.656716	1.34	360.00	2341.20	268.656716	1.364217	366.506024	2900.866248
6	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1.75L	185.589744	210.60	39085.20	99027.48	185.589744	110.560063	20518.813797	44989.509807
7	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 750M	222.360000	155.00	34465.80	99027.48	222.360000	41.041148	9125.909702	44989.509807
8	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1L	230.786122	110.39	25476.48	99027.48	230.786122	66.489207	15344.786307	44989.509807
9	SPECS	JACK DANIELS BLK WHSKY 1L	229.533835	476.14	109290.24	157421.22	229.533835	217.722084	49974.584892	63726.848507
10	SPECS	JACK DANIELS BLK WHSKY 1.75L	185.589744	176.67	32788.14	157421.22	185.589744	44.962771	8344.629157	63726.848507
11	SPECS	JACK DANIELS BLK WHSKY 750M	222.360000	69.00	15342.84	157421.22	222.360000	24.319277	5407.634458	63726.848507

In [31]:

```
full_opt_stats[['Chain Master','Product'
               ,'P0 Optimal Price','P0 Demand','P0 Revenue','P0 Chain Master Revenue'
               ,'P0+Sim Optimal Price','P0+Sim Demand','P0+Sim Revenue','P0+Sim Chain Master Revenue'
               ]]
```

Out[31]:

	Chain Master	Product	P0 Optimal Price	P0 Demand	P0 Revenue	P0 Chain Master Revenue	P0+Sim Optimal Price	P0+Sim Demand	P0+Sim Revenue	P0+Sim Chain Master Revenue
0		JACK DANIELS BLK WHSKY 1L	227.71	296.250933	67459.300020	139577.0	228.75	208.300856	47648.820791	121133.0
1		JACK DANIELS BLK WHSKY 1.75L	184.44	184.916178	34105.939952	139577.0	169.93	244.776787	41594.919344	121133.0
2		JACK DANIELS BLK WHSKY 750M	223.78	169.864344	38012.242857	139577.0	224.13	142.278525	31888.885832	121133.0
3	THE BARREL HOUSE	JACK DANIELS BLK WHSKY 1L	222.56	225.475606	50181.850796	50962.0	234.28	1.199191	280.946564	1059.0
4	THE BARREL HOUSE	GENTLEMAN JACK WHSKY 6PK 1L	295.95	1.484805	439.428095	50962.0	296.93	1.484796	440.880342	1059.0
5	THE BARREL HOUSE	JACK DANIELS BLK WHSKY LSE 50M	267.99	1.272531	341.025472	50962.0	268.45	1.255788	337.116310	1059.0
6	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1.75L	191.28	166.023589	31756.992175	80824.0	185.37	552.712908	102456.391798	123698.0
7	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 750M	217.61	165.010479	35907.930285	80824.0	217.87	26.238294	5716.537039	123698.0
8	WESTERN BEV LIQ TX	JACK DANIELS BLK WHSKY 1L	224.24	58.682795	13159.029857	80824.0	214.47	72.389591	15525.395586	123698.0
9	SPECS	JACK DANIELS BLK WHSKY 1L	226.50	222.317808	50354.983567	82561.0	229.22	246.414206	56483.064251	66969.0
10	SPECS	JACK DANIELS BLK WHSKY 1.75L	184.04	49.283717	9070.175337	82561.0	181.38	31.088421	5638.817880	66969.0
11	SPECS	JACK DANIELS BLK WHSKY 750M	205.54	112.562539	23136.104353	82561.0	209.29	23.160338	4847.227201	66969.0

In [ ]: